ECE 445

SENIOR DESIGN LABORATORY

FINAL REPORT

Display of Partial Differential Equations

Team #5

KEJIA HU
(kejiahu2@illinois.edu)
ZHUOHAO LI
(zhuohao5@illinois.edu)
QIANHE YE
(qianhey2@illinois.edu)
QIRONG XIA
(qirongx2@illinois.edu)

Advisor: Prof. Pavel Loskot
TA: Yue Yu

2024-05-10

Abstract

This project developed a 3D visualization system for accurately representing the solution of 2D partial differential equations. User can choose different partial differential equations and different boundary conditions, and then our Raspberry Pi Pico will solve the equation and control the movements of the sticks. Ultrasonic sensors are implemented to get the current heights of each sticks. The feedback from sensors made the display more accurate. Finally, a canvas is put on the top of the sticks and a projector is projecting different colors from the top for a better visualization. As a whole, this device can show users a intuitive solution of the partial differential equation that they choose.

Contents

1 Introduction

1.1 Background

Understanding the behavior of complex systems in scientific and engineering fields often requires analyzing interactions between multiple variables over time and space. While three-dimensional (3D) visualization offers superior expressiveness and effectiveness in representing data information [1]. Modern software capabilities include sophisticated 3D visualization tools that not only enhance data comprehension but also facilitate interactive exploration and analysis [2]. However, the current screen-based 3D visualization tools are confined to the dimensions of the display device, hindering the viewer's ability to perceive objects accurately from various angles and distances. There's a pressing need for innovative visualization tools capable of representing complex systems in 3D in real-time, providing a more intuitive understanding of their behavior. Such tools could find applications in various disciplines, from mathematics to physics, biology, engineering, and environmental science.

1.2 Solution Overview

To address the challenges posed by traditional visualization methods, our solution proposes the development of a user-friendly 3D visualization system for accurately representing 2D differential equations. The system will dynamically visualize the changing behavior of the function over time, projecting color onto the surface from the top to enhance interpretation. It comprises four subsystems: User Interface subsystem, Control subsystem, Mechanical subsystem, and Coloring subsystem. The User Interface subsystem collects the user's differential equation input and transmits it to the Control system. The Control subsystem converts this information for mechanical devices to understand and adjust the height of sticks, creating a smooth visualization surface. Finally, the Coloring Subsystem projects different colors onto the canvas. A visualization of the device is shown in Fig 1. This solution not only improves data comprehension but also facilitates interactive exploration and analysis, offering researchers a comprehensive tool for understanding complex systems' behavior.

1.3 Block Level Changes

The final version of the block diagram is shown in Fig 2.

- User Interface Subsystem: There is no block-level change in the User Interface Subsystem.
- Coloring Subsystem: Instead of connecting the coloring subsystem to the control system, we connect it to the use interface subsystem. Because Raspberry Pi Pico does not have a mature tool for generating color plots, which will be used in the Coloring Subsystem.
- Control Subsystem: Initially we use two development boards to control the 16 mo-

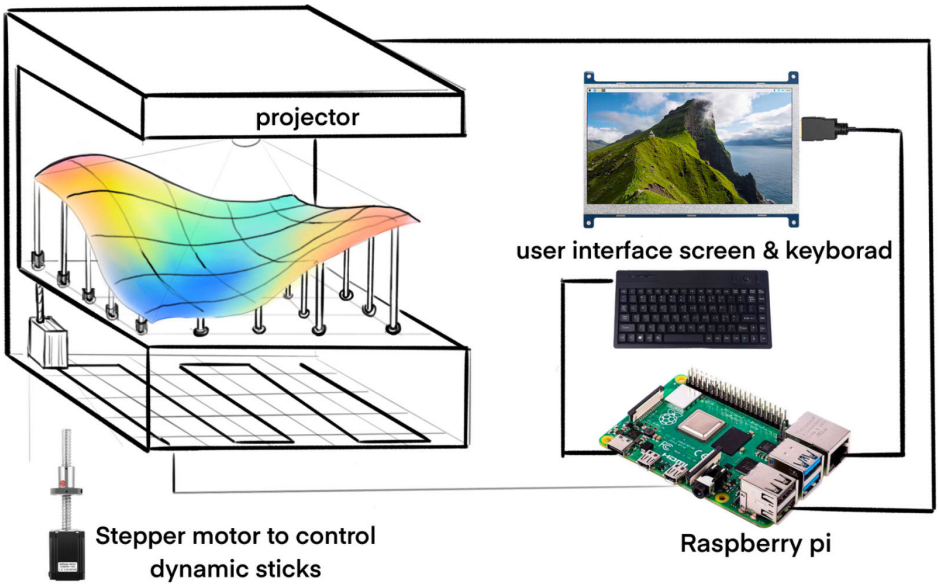


Figure 1: A pictorial representation of the device

tors, in which the motor driving part is replaced by a PCB board in the later versions. The type of sensors also changes from the ultrasonic sensors to the infrared sensors.

- Mechanical Subsystem: On the foundation of the original simple design, sensors and their bases were added to the mechanical subsystem to measure the height of rods.

2 Design

2.1 Design Procedure

2.1.1 User Interface

The User Interface (UI) Subsystem is an essential component that bridges the gap between users and the visualization system, enhancing the interaction through a well-designed, user-friendly graphical interface. Through user interface subsystem, user can choose a partial differential equation and boundary conditions. The choice will be sent to the Raspberry Pi pico. The Raspberry Pi Pico will solve the equations and transfer the solution back to the laptop. The solution information will be used for the coloring subsystem. To transfer data back and forth between the laptop and Raspberry Pi Pico. We considered two methods, C and Python.

C Language uses Universal Asynchronous Receiver-Transmitter (UART) which enables efficient and fast direct hardware-level communication. And it's suitable for real-time applications when the timing of transferring data is critical. However, it's more complex to implement and debug.

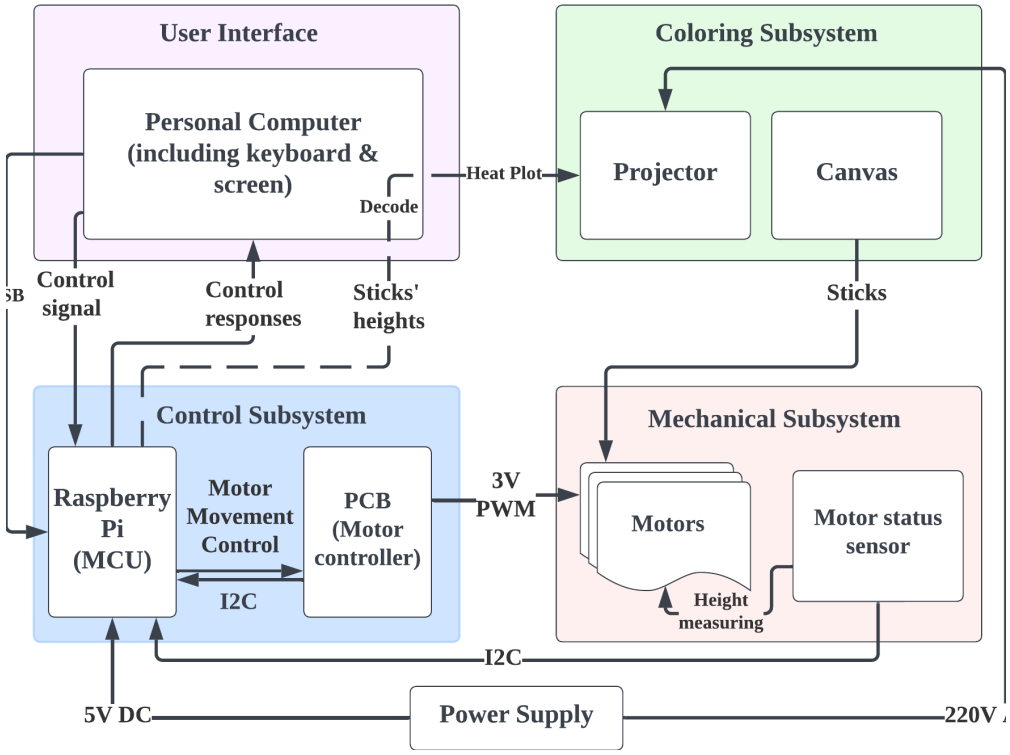


Figure 2: Block Diagram for design

Python uses Serial Communication. It has high-level and easy-to-use API provided by libraries like $PySerial$. Meanwhile its cross-platform compatibility making it easy to port code between different systems. Nevertheless, this method transfers data relatively slower compared to direct hardware-level communication like UART in C. So it might not be suitable for real-time applications with strict timing requirements.

Due to the limitation of the Rotation per Minute (RPM) of our motor, we don't require our device to be real-time. Plus, we implement our Control subsystem using Python. Therefore, we decided to use Python for data communication and user interface.

Finite Difference Method (FDM) is used to solve the PDEs numerically. It's a common method for solving differential equations, particularly partial differential equations (PDEs), by discretizing the spatial and/or temporal domain into a grid of discrete points. This method replaces derivatives in the differential equations with finite difference approximations, turning the differential equations into algebraic equations that can be solved numerically. Approximating the fist and second derivative as follows:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{1}$$

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{2}$$

By replacing the derivatives with difference approximations, we are able to solve the PDEs numerically.

2.1.2 Coloring Subsystem

Before the actual implementation of the Coloring subsystem, we think of two ways to achieve canvas coloring. One is to use the Bluetooth module on the Raspberry Pi Pico W to send heat plots to the projector for display, and the other way is to use the personal computer connecting to the projector. After some exploration of Pico, we concluded that Pico does not have a mature tool for generating heat maps for further display. Therefore, we decided to integrate the User Interface Design 2.1.1 with the Coloring Subsystem. The projector is directly connected to the laptop, and the laptop should be able to receive signals from the Pico, decode the signals, and generate a heat map that is submitted to the projector via an HDMI cable. The projector is vertically above the canvas so that we do not need to rectify the displayed images.

2.1.3 Control Subsystem

Motor Types We initially considered using stepper motors and angular displacement sensors to track the actual state of the motor motion. However, due to budget constraints, we were unable to utilize the advanced features offered by stepper motors and opted for a screw motor instead.

Motor Drivers Initially, the 16 motors in our system are controlled by two Raspberry Pi Pico boards as the number of pins on one board is limited for controlling 16 sensors simultaneously. The motors and the development boards are connected to the breadboard, which turned out to be unstable due to various reasons such as the low current provided by the Pico that cannot meet the requirement for motors' movement. We also found that connecting via breadboard makes poor contact between the circuit components, further increasing the instability of the system. Even though we have successfully enabled communication between two boards by connecting the submission ring (the hardware-assist queue) of one board to the receiving ring of the other board, the motors still fail to move sometimes.

In our second attempt, the motors are connected to the output latches from two I^2C expansion chips (type MCP23017), which are connected to one of the Raspberry Pi Pico I^2C channels. By receiving the control signals from the MCU, we are assuming that the 32 GPIO ports on the chips can provide power to the motors. However, the current supplied by the expansion chip's output ports is 7mA to 10mA, smaller than the working current of motors.

Due to all these reasons above, we finally decided to use a Printed Circuit Board (PCB) as a motor driver, to control the movements of all 16 motors. The design of the PCB has gone through several modifications, the final version of the PCB has three major parts: (1) the motor drivers, (2) the I^2C expansion chips, and (3) the voltage regulator, as illustrated in Fig 4.

Distance Measuring Sensors Initially, we considered using ultrasonic sensors for distance measurement. However, ultrasonic sensors require multiple pins per unit, making it impractical to connect multiple sensors directly to a single Raspberry Pi Pico due to its limited number of GPIO pins. This limitation led us to explore other sensor types.

We then evaluated the use of infrared sensors, which can be connected via the I2C communication protocol. I2C allows for multiple sensors to be connected to a single port; however, we faced a challenge with address conflicts as all the sensors shared the same I2C address and could not be differentiated on a single I2C bus.

To resolve the addressing issue without increasing the number of required I2C pins, we decided to implement multiplexers (MUXes). By using a MUX, we could connect multiple sensors with identical addresses to different channels of the MUX, thereby isolating them effectively. This setup allowed us to connect up to eight sensors per MUX.

Given our requirement to utilize more than eight sensors, we opted to use two MUXes. Each MUX was configured with a unique I2C address and connected to eight sensors. This configuration enabled us to manage 16 sensors in total using a single I2C bus on the Raspberry Pi Pico.

The final design comprises a single I2C bus on the Raspberry Pi Pico connected to two multiplexers. Each multiplexer interfaces with eight infrared sensors. During operation, sensor data is collected via the I2C bus, which is then processed by the Pico to determine the necessary adjustments for motor control. This design efficiently handles the sensor data with minimal latency and uses the Pico's limited GPIO resources effectively.

2.1.4 Mechanical Subsystem

The Mechanical Subsystem consists of rods that can move up and down dynamically in accordance with the displayed solution. Each rod is connected to a motor, which is fixed in a base and aligned on a board, forming a 4×4 grid. The motor and a screw was used to transfer rotation motion into translation, and the base for both motor and sensor was created by CAD modeling and 3D printing. The detailed design is shown below. A layer of silk stocking-like polyester fiber is secured to the top of each rod, ensuring the visualization remains seamless. Inputs from the control subsystem dictate the movements of this subsystem, which then executes these commands. Additionally, it provides feedback to the control subsystem for adjustments and fine-tuning with sensors. The system is designed to adhere precisely to the commands from the control system, aiming for high accuracy and precision in movements. It's built to be sturdy, effectively reducing minor vibrations and environmental noise. In addition, to support the projector in Coloring Subsystem, a shelf is used.

2.2 Design Details

2.2.1 User Interface

The implementation for data transfer between the laptop and Raspberry Pi Pico involves bidirectional serial communication [3]. On the laptop side, the code initializes a serial connection with specified parameters including port and baud rate. Data is then encoded as strings and sent over this serial connection to the Raspberry Pi Pico. On the Raspberry Pi Pico side, the code continuously listens for input from the standard input (stdin), reading characters until data reception is complete. After the Raspberry Pi Pico calculates the solution of the indicated PDEs, it transmits the solution data over the serial connection back to the laptop. The code for data transmission is shown in Appendix D.

Partial Differential Equations chosen for display in our projects are the 2D Heat Equation, Wave Equation, and Laplace Equation. We use Equation 1 and 2 to solve the PDEs numerically.

2D Heat Equation:

$$\frac{\partial u}{\partial t} = k\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{3}$$

Solving the equation as:

$$u_{i,j}^{n+1} = u_{i,j}^{n} + k\Delta t\left(\frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{(\Delta x)^2} + \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{(\Delta y)^2}\right) \tag{4}$$

2D Wave Equation:

$$\frac{\partial^2 u}{\partial t^2} = c^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{5}$$

Solving the equation as:

$$u_{i,j}^{n+1} = 2u_{i,j}^{n} - u_{i,j}^{n-1} + c^2\Delta t^2\left(\frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{(\Delta x)^2} + \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{(\Delta y)^2}\right) \tag{6}$$

2D Laplace Equation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{7}$$

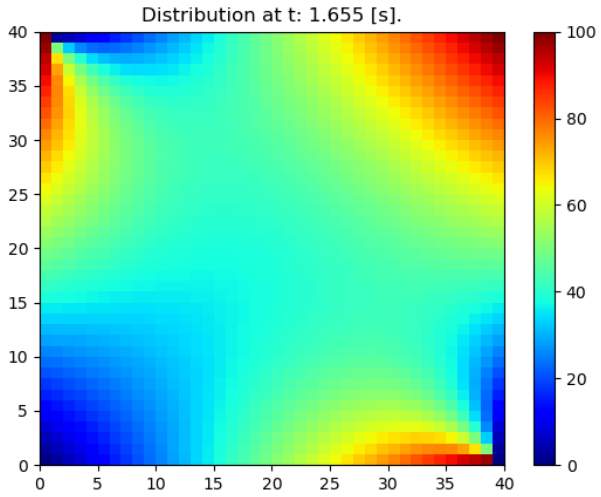
Solving the equation as:

$$u_{i,j}^{n+1} = \frac{1}{4}(u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}) \tag{8}$$

2.2.2 Coloring Subsystem

Color Projection In Sec 2.2.1, we transfer the solution from the Pico to our personal computer and stored the data in a file which facilitate our Coloring subsystem. Given the memory constraints of the Raspberry Pi Pico, we address 2D partial differential equations by sampling 25×25 points. For a smoother visualization of the color plot, we do linear interpolation to increase the sampling points from 25×25 to 100×100.

To make the projector vertically above the canvas, we used a vertical cloth hanger and fixed the projector on the top. An illustration of the color plot and the projector configuration is shown in Fig 3



(a) Example of the projected color for the heat function

(b) Projector Configuration

Figure 3: The illustrations of coloring subsystem

2.2.3 Control Subsystem

Microcontroller We use the Raspberry Pi Pico as our microcontroller. Unlike other boards developed by the Raspberry Pi Foundation, Pico cannot be installed with an operating system, making it suitable for the control of lightweight electronic projects. The hardware parameters of the Raspberry Pi Pico are shown in Tab 1. In our project, Pico is the core controller, interacting with the UI subsystem, calculating the solutions of the PDE, analyzing the feedback from the sensors, and integrating all the information to control the movements of the motors.

Motor Drivers We incorporate three parts in our PCB, the voltage regulator, the motor drivers, and the I^2C expansion chips. The major parts of the circuit are shown in Fig 5, and the overall circuit schematic and the PCB design are shown in Appendix A.

- **Voltage regulator**: The voltage regulator is the power supply unit for the whole PCB. It consists of an L78M05 chip, which can take the input voltage ranging from

Components	Values
CPU	Dual-core Arm Cortex-M0+ Processor
Clock freq	133MHz
Memory/storage	264kB SRAM
	2MB on-board flash memory

Table 1: The built-in parameters of the microcontroller

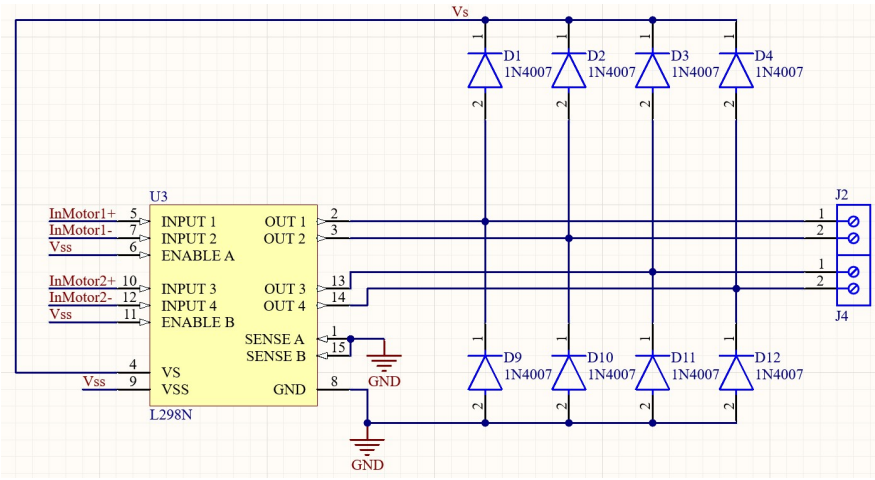
5V to 12V. The two capacitors connecting to the input and the output sides of the L78M05 chip serve as the buffer for the sudden change of voltage and thus provide more stability to the power supply.

- **Motor drivers**: Each motor driver unit can drive two motors, providing stable power to the motors. Since there are 16 motors in our device, we duplicate the motor driver unit 8 times in the circuit. The dual full-bridge motor driving chip with type L298N is used. It takes `Vs` as the power supply for the chip and `Vss` as the logic supply voltage. The PWM signals for the two motors are passed to the `INPUTx` where `x` is the integer from 1 to 4. The `OUTPUTx` is the output signal connecting to two motors, where `x` is the integer ranging from 1 to 4.
- **I^2C expansion chip**: The two I^2C expansion chips serve as a bridge between the microcontroller and the mechanical parts. We choose the MCP23017 as the I^2C expansion chip so that the 16 motors can be controlled by the microcontroller using only two pins. The internal architecture of the MCP23017 chip is shown in Fig 5. The Raspberry Pi Pico can send data to the expansion chips by specifying the address of the chip. Every time the Pico will send two bytes to the chip, which is stored in the 16 output registers on the chip. The output ports of the I^2C expansion chips are connected to the motor drivers' inputs.

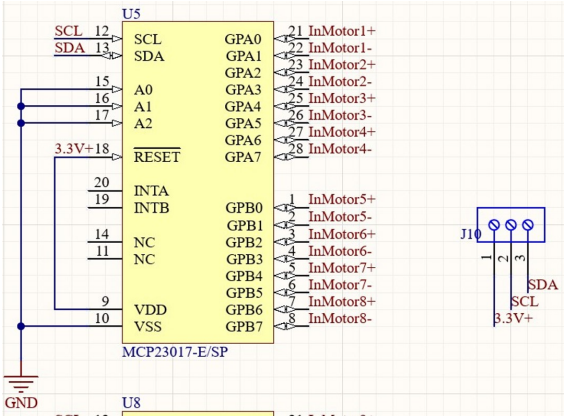
In conclusion, the PCB features logic input ports that receive the SDA and SCL signals from the Raspberry Pi Pico microcontroller. These signals are then processed through an expansion chip, which converts them into signals for motor movement. The PCB's power supply is connected to an existing power module, which can utilize a USB port on a laptop to deliver a 5V output.

Distance Measuring Sensors For our project, we have utilized the I2C0 bus on the Raspberry Pi Pico as the primary communication channel for sensor data. This bus utilizes pin12 (SDA) and pin13 (SCL) as the I2C port. We have integrated two multiplexers (PCA9548A) into the system, assigned with addresses 0x71 and 0x70 respectively, to facilitate the connection of multiple sensors without address conflict.

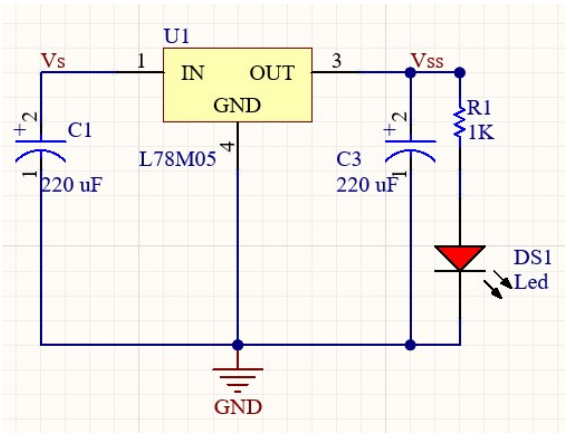
Each PCA9548A multiplexer offers 8 I2C channels, enabling us to manage multiple sensors that inherently share the same default I2C address (0x29). By selectively enabling and disabling individual channels on the multiplexers, we are able to isolate and commu-



(a) The motor driver



(b) The I^{2}C expansion chip



(c) The voltage regulator

Figure 4: The circuit schematic

nicate with each sensor independently.

The sensors used in this design are the VL53L0X, which are time-of-flight (ToF) ranging sensors. We initialize 16 instances of these sensors corresponding to each channel of the multiplexers. The distance measurement process involves sequentially pinging each sensor to obtain distance readings. This is performed in a cycle where each sensor is accessed one after the other, collecting distance data serially.

Once a complete cycle of distance measurements is accomplished, the collected data array is transmitted to the motor driver. This enables the motors to adjust their operation based on the distance measurements received. The loop of measurement and motor adjustment continues until all sensor readings align with the predetermined target values, as defined by the system's design equations.

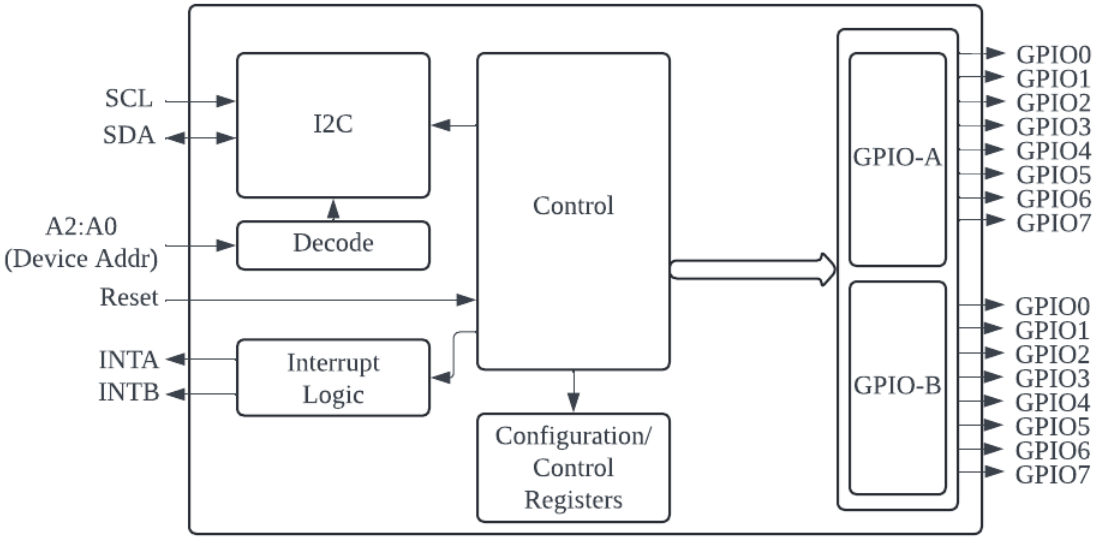


Figure 5: Internal Architecture of MCP23017

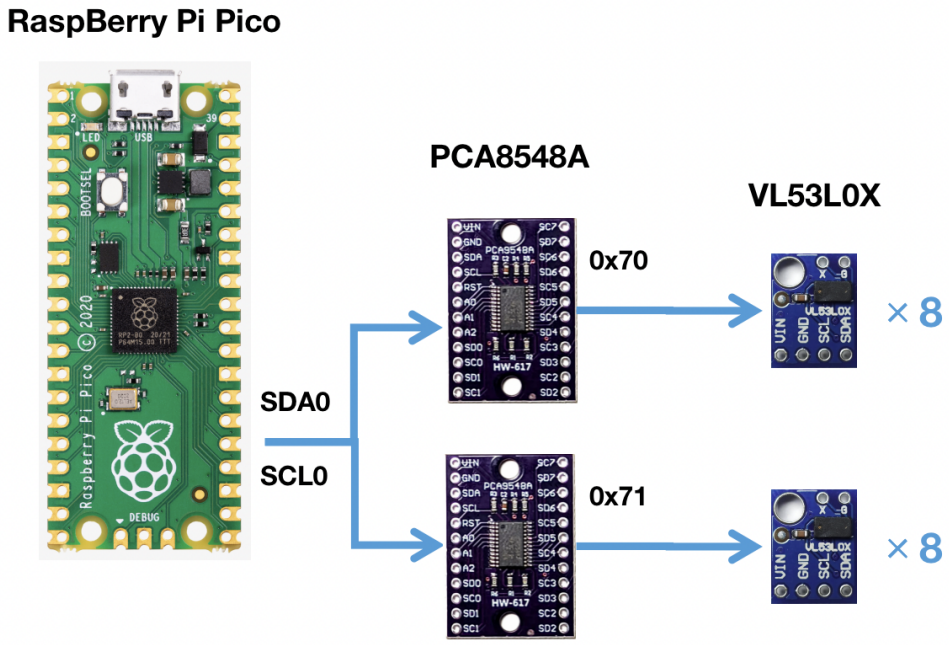


Figure 6: Sensor Design

2.2.4 Mechanical Subsystem

Transmission mechanism Our project of visualization of PDEs features the accurate representation of the relative position of different points on the solution of specific functions, which is realized by the controlled height of several sticks. In order to convert rotation to translation, the method of using a flange nut and screw was chosen. As shown in the

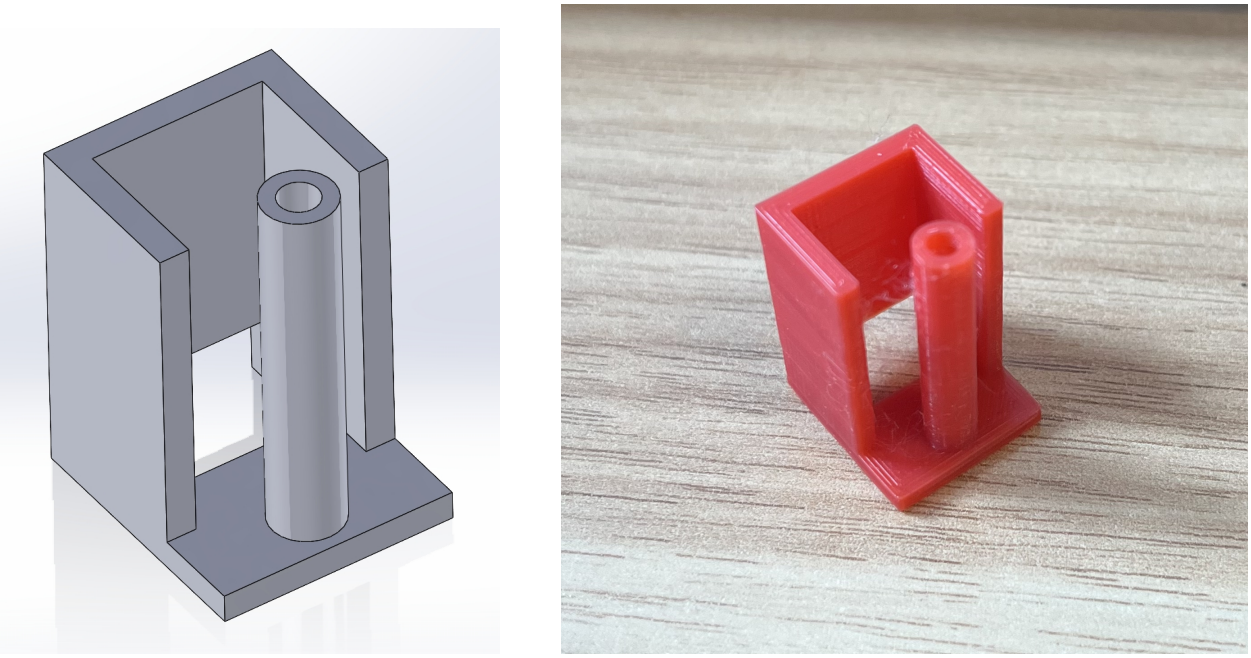
Fig Fig 7, when the motor operates, the screw rotates along with the motor. If the nut is restricted from spinning, it will move up and down according to the rotating direction of the screw, i.e. the motor. Since the Raspberry Pi Pico was chosen as the development board to control and route the signal as well as provide power, the motor chosen is powered by 3V. Also, considering the trade-off between torque and rotation speed, the speed of 500rpm was chosen.

Figure 7: The motor and screw

CAD modeled and 3D printed parts

- **Rod** The end of the rod is attached to the canvas above so that the specific point of the canvas can be moved up and down with the rod. The rod should not be interfered with the screw that is attached to the motor, and be as stable as possible when translating. The rod is hollow inside to avoid collision with the screw, and the end of it is flat in order to connect to the canvas. Another end of the stick is designed to connect with the flange nut using a nut and bolt.
- **Motor base** To form a 4×4 grid of solution points, the motors must be standing straight up and fixed onto a baseboard. The motor base was designed via Solid Works and 3D printed. As shown in Fig 8, the motor could be vertically inserted into the base, and the vacancy behind was designed to let the wires soldered on the motor pass through; a steel stick with a length of 10 cm could be inserted into the hollow cylinder at the front, which serves to constrain the rotational motion of nut. The length of the steel stick was carefully chosen to be no longer than the height of the top of the rod when it is at its lowest position, as well as providing the restriction of rotation to the end of the screw.
- **Sensor base:** Each motor requires a sensor to measure the position of the rod, so there are 16 sensors in total. As shown in Fig 9, the sensor base could hold the sensor to let the infrared module face upward, and the vacancy behind was designed to let the wires inserted pass through. The sensor base is designed to be as low as possible since the sensor does not perform well when it gets too close to the obstacle.

Assembly For the assembly, both the motor and stick are inserted into the motor base, and the rod is connected to the flange nut. The assembled body is shown in Fig 10. During the



(a) 3D model

(b) Printed version

Figure 8: Motor base

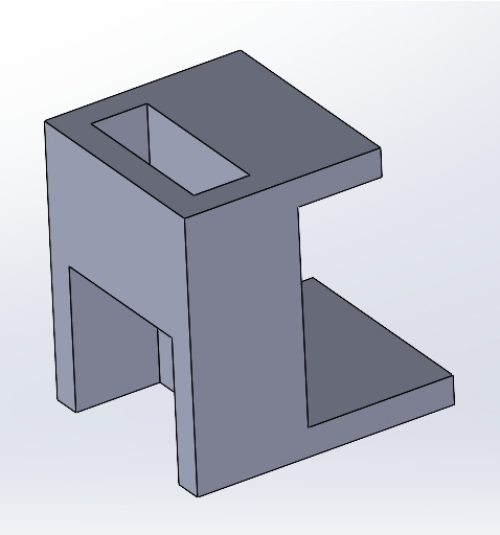
operation of motor, the flange nut would move along the stick, and its maximum travel distance is determined by the length of the stick inserted in the cylinder.

Layout In order to represent the solution of functions relatively accurate as well as considering the complexity, the capacity of development board and the budget, we decided on the 4×4 grid of controlled units shown in Fig 11. Each unit is 7.5cm apart from the other, and they are glued onto an acrylic board.

Canvas The canvas material on top of the rods that we chose was the silk stocking-like polyester fiber because it has a low elasticity coefficient. It also has the color of white, which makes it suitable for the projection of color. Previously, we have tried the silicone pads, but the motors have no enough torque to pull the material apart. The final look is shown in Fig 12.

Wood chip As shown in Fig 13, laser cut was used to create 16 pieces of wood chip. Each of the chips was secured somewhere in the middle of the rod to serve as an obstacle to the sensor. When the rod moves, the distance between the chip and sensor changes, which gives feedback to control the motor motion. The position of the chip does not matter, as long as it is on top of the infrared module, since the control code provides an offset for each sensor.

Shelf As shown in Fig 14, the shelf has a width of 1m and a height of 1.2m. With the help of a Projector hanging bracket and tape, the projector can be hung on the shelf and project color downward onto the canvas.



(a) 3D model

(b) Printed and assembled version

Figure 9: Sensor base

3 Verification

3.1 User Interface

3.1.1 Verification of User Interface

We verify that users can interact with the graphical interface as intended. Fig 15 shows the way that user select the partial differential equations and boundary conditions. We ensure that the chosen equations are successfully transmitted to the Raspberry Pi Pico. By comparing the result in Pico and data in our laptop, we confirm that the solution data is received back from the Pico and displayed appropriately.

```
Choose the partial differential equation type and boundary conditions:
1. Heat equation -- Fixed temperature at boundaries (0°C on left and bottom, 100°C on right and top)
2. Heat equation -- Linear boundaries (linear temperature gradient from 0°C to 100°C along each edge)
3. Wave equation -- Source term at upper left side
4. Wave equation -- Source term at lower right side
5. Wave equation -- Source term at upper left side with smaller radius
6. Laplace equation -- f(x) = sin(pi*x) at left boundary
7. Laplace equation -- f(x) = x^2 at left boundary
8. Laplace equation -- f(x) = exp(-x) at the left boundary
Enter your choice (a number from 1 to 8): 6
```

Figure 15: User Interface

Figure 10: Assembly of one transmission unit

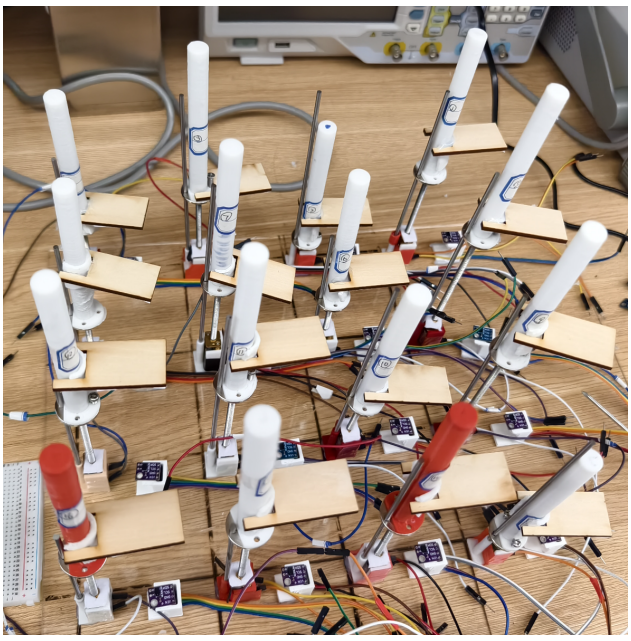


Figure 11: 4×4 grid on board

3.1.2 Verification of PDE Solving

To verify that the Finite Difference Method (FDM) works, we compare the results solved numerically using FDM with the analytical solution. However, it's hard to find an ana-

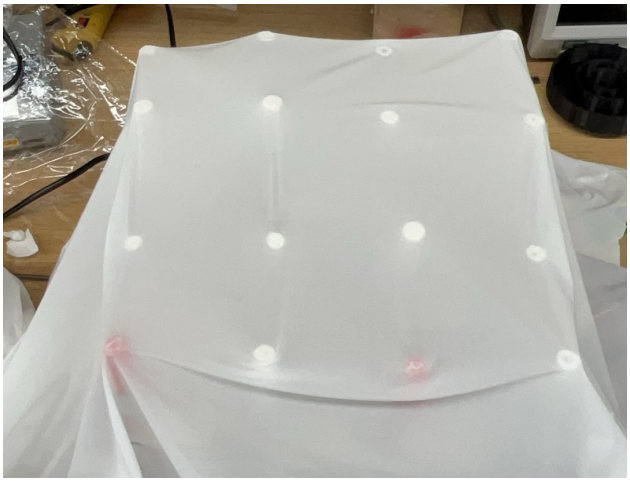


Figure 12: The whole assembly covered by canvas

Figure 13: Wood chip

lytic solution for every PDE. Inspired by [4], we explored the method of manufactured solutions to get an idea of how accurate is our FDM method. The 2D Poisson equation is given by:

$$\nabla^2 u(x, y) = f(x, y) \tag{9}$$

where $u(x, y)$ is the unknown function to be solved for, and $f(x, y)$ is the source term.

We choose the exact solution to be the 2D Poisson equation is given by:

$$u(x, y) = \sin(\pi x) \sin(\pi y) \tag{10}$$

where x and y are the spatial coordinates in the domain. Using Equation 9 and 10, we can get:

$$f(x, y) = -2\pi^2 \sin(\pi x) \sin(\pi y) \tag{11}$$

To solve the 2D Poisson equation numerically using FDM, we discretize the domain into a grid of points. Let $u_{i,j}$ represent the numerical approximation to the solution at grid point

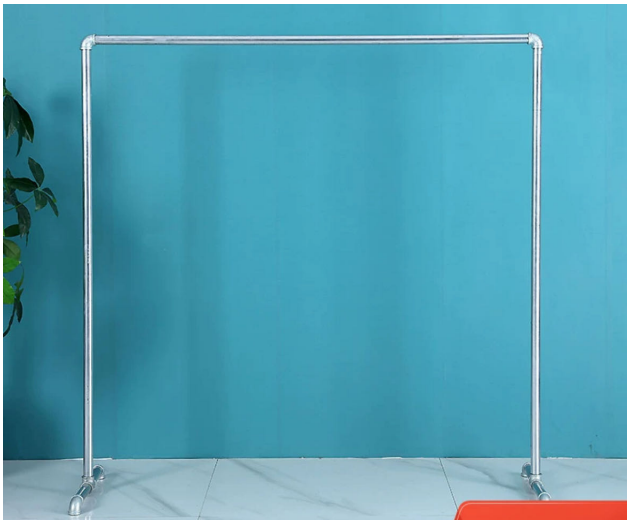


Figure 14: Shelf

(x_i, y_j). Using central difference approximation for the second derivatives, the discretized form of the Poisson equation is given by:

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta x^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta y^2} = f_{i,j} \tag{12}$$

where $f_{i,j}$ represents the source term at grid point (x_i, y_j), and Δx and Δy are the grid spacings in the x and y directions respectively. We assume x and y are the same here.

Solving the equation as:

$$u_{i,j}^{n+1} = \frac{1}{4}(u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n - \Delta x^2 f_{i,j}^n) \tag{13}$$

This equation is solved iteratively. The process involves updating each grid point based on the values of its neighboring grid points until a convergence criterion is met.

Eventually, we get the Mean Square Error (MSE) to be 3.86×10^{-5} and a comparison of the numerical solution and the exact solution is shown in Fig 16. We can verify our FDM method to get the solutions of PDEs is accurate.

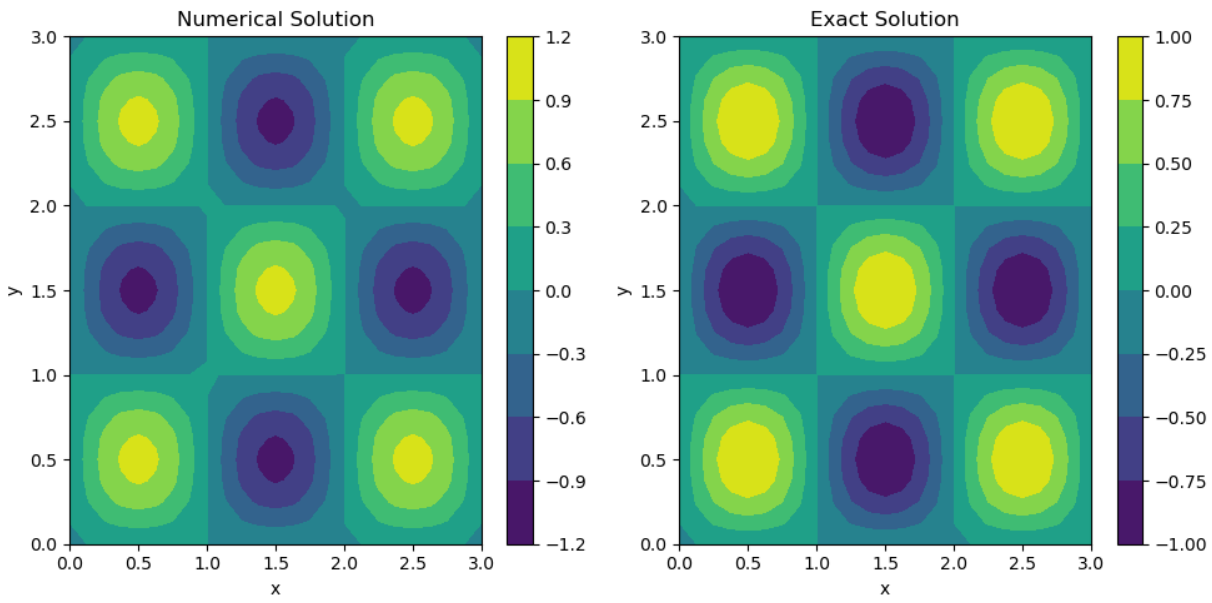


Figure 16: A comparison of numerical solution and exact solution

3.2 Coloring Subsystem

The Coloring subsystem accurately interprets the solution data received from the UI subsystem. And the generation of heat maps based on the solution data is accurate.

Canvas Material Choice In order to project colors onto sticks of varying heights accurately, it's essential to place a stretchable canvas atop the sticks. This material enables the canvas to accurately represent the peaks and troughs of the curve. We considered different materials:

- **Flannel** is a soft texture with good absorbency. However, it has limited stretchability and may not conform well to irregular shapes or curves. Also, it's susceptible to wrinkling and creasing, which might affect the accuracy of the representation.
- **Rubber** has high elasticity, allowing for excellent stretchability. It is also durable and resistant to tearing, making it long-lasting. However, it's a stiff texture that requires a large force to stretch this material. Our motors are not powerful enough to stretch rubber effectively.
- **Spandex** has exceptional stretchability, ensuring a snug fit over irregular shapes and curves. The motor can easily change the shape of spandex because of its lightweight. And it retains its shape well over time, maintaining the accuracy of the representation.

After trials, we decided to use spandex as the canvas.

3.3 Control Subsystem

3.3.1 Verification of motor driver

According to the motor's datasheet, the RPM without workload is supposed to be 500, while the RPM is reduced to 400 with workloads. The working current without workloads is 20 mA and 30 mA with workloads. Considering the case when the motor is blocked and stops rotating, the output current can be up to 100 mA, with the torque 0.15 $kg \cdot cm$.

we measure the actual voltage and the current values outputted by a PCB using a voltmeter and an ammeter. In our case, we only focus on the tests with workloads on the motors. The measurement results are shown in Tab 2. The actual voltage of the motor is 0.5V lower than expected and the current is 4mA lower than expected. The inaccuracies introduced by the PCB design are represented in the sticks' linear speed. Therefore, we measure the moving distance and the actual moving duration of eight sticks, which is shown in 17. In this experiment, we let the sticks move from the lowest position to their highest position and measured the completed time accordingly. The average moving distance is 10 cm, and the linear speeds range from 0.391 cm/s to 0.669 cm/s. Given the screw leas is 0.5 mm/rev, the RPM ranges from 469 to 802.

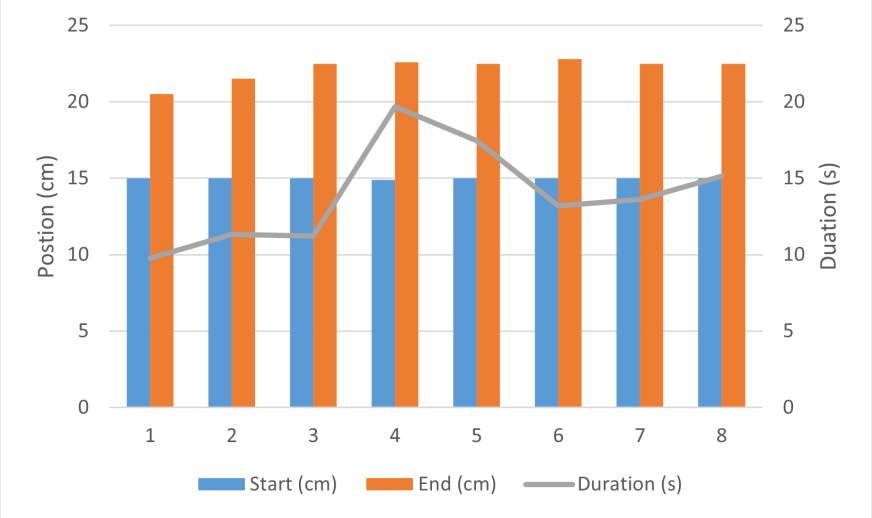


Figure 17: Moving distance and the completion duration

3.3.2 Verification of stick height measuring

We use distance-measuring sensors to capture the height of the sticks. The response latency of the 16 sensors is shown in Fig 18. The average duration between the microcontroller sending the read request to the sensor and receiving the request is 620 μs. Therefore, iterating all the 16 sensors takes at least 9.92 ms. Upon receiving the sensor's response, the microcontroller will send a request to the corresponding motor's movement. For a motor, the average period of (1) reading from the sensor, and (2) sending a request to the motor is 10 ms. In this period, the motor's moving distance is at least

Motor Index	Voltage	Working current	Working power
1	2.46V	0.015A	0.0369W
2	3.76V	0.017A	0.0639W
3	2.42V	0.016A	0.0387W
4	2.43V	0.021A	0.0510W
5	2.38V	0.018A	0.0428W
6	2.42V	0.019A	0.0460W
7	2.32V	0.015A	0.0348W
8	2.25V	0.015A	0.0338W
9	2.38V	0.014A	0.0333W
10	2.45V	0.016A	0.0392W
11	2.49V	0.018A	0.0448W
12	2.44V	0.017A	0.0415W
13	2.28V	0.016A	0.0365W
14	2.33V	0.014A	0.0326W
15	2.47V	0.015A	0.0371W
16	2.34V	0.014A	0.0328W
Average	2.47V	0.016A	0.0395W

Table 2: The working power and current of the 16 motors

$0.391cm/s \times 10ms = 0.00391cm$, meaning that the electronic control is far quicker than the mechanical movements, and achieves the control in small granularity.

3.4 Mechanical Subsystem

3.4.1 Choice of motor

There are several types of 3V motors with rotation speeds ranging from 10 rpm to 3000 rpm. With the higher speed, the stick could go to the assigned position more quickly, but with the risk of being stuck because of the inadequate torque. During our test of different types of motors, we found out that the 500 rpm one is just enough for securing the rotation as well as guaranteeing relatively high speed.

3.4.2 Optimization of 3D printed parts

Due to the inaccuracy nature of 3D printing and the small scale of the motor, the design of parts needs several times of revisions. For the motor base, the first version's space to place the motor is too large, which makes the motor wobble inside the base during the operation, and the wall of the cylinder to place the steel rod is too thin and cannot

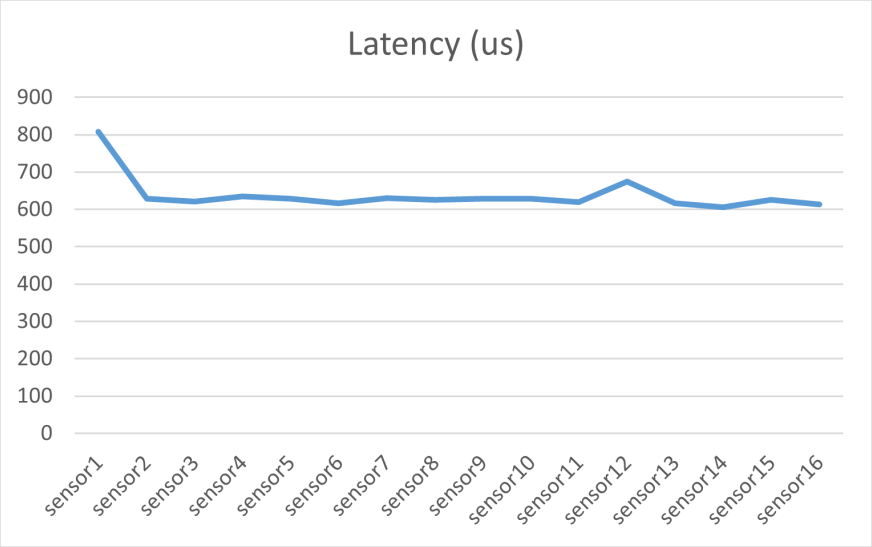


Figure 18: Sensors' response time

be printed out. In the second version, I fixed these problems and the motor base works fine. For the rod on top of the flange nut, the first version has holes that are too close to the stick, or the stick is too thick so that the bolt cannot be placed. Also, those holes are not large enough to insert a shaft. In the second version, the holes' size is adjusted, but because of the limited space, it is still difficult to place the bolt, but the design is usable. After optimization of these parts, the motor could stand stably during the whole operation without noticeable wobbling, and the rod is firmly attached to the flange nut. For the sensor base, the final version ensures the sensor is held as close as possible to the baseboard to ensure functionality.

4 Costs

4.1 Labor Costs

According to public statistics of the average salary for a graduate from Illinois ECE [5], the average salary per hour is roughly \$45/hour, which is 315 RMB/hour. Assuming that we are given 9 weeks to finish the project, every member works for at least 20 hours per week, the total number of hours for each member is given by

$$20 \text{ hours/week} \times 9 \text{ weeks} = 180 \text{ hours}$$

Multiplying the total number of hours worked by the hourly rate gives the total labor cost for a person, which is

$$315 \text{ RMB/hour} \times 180 \text{ hours} = 56700 \text{ RMB}$$

Therefore, the total labor cost for this project will be

$$56700 \text{ RMB/person} \times 4 \text{ persons} = 226800 \text{ RMB}$$

4.2 Electronic Components Costs

The cost for each component for our project is given in Tab 3.

Components	Vendor	Quantity	Cost (RMB)/unit	Total Cost (RMB)
DC Motor	YongChuangXin Actuator	17	22	374
Raspberry Pi Pico	Raspberry Pi	2	31	62
Raspberry Pi Pico W	Raspberry Pi	2	59	118
I2C Expander Chip	Microchip	2	9.5	19
Sensors	Xintai Microelectronics	18	5.9	106.2
Components on the PCB	Various vendors	Multiple	95.18	95.18
PCB	Jialichuang	1	10	10
Projector	Jiying	1	438	438
HDMI Cable	Hantangke	1	9.85	9.85
Canvas (Flannel)	Shengshi Textile and Leather	1	6.5	6.5
Canvas (Rubber)	Chunshi	1	3.7	3.7
Canvas (Spandex)	Boya	1	30	30
Projector stands	Tengtai cloth stands	1	121	121
Projector Power Supply Cable	Shanze	1	19.8	19.8
			Total	1413.23

Table 3: Cost for components

5 Conclusions

5.1 Accomplishment

In conclusion, our efforts lead to the development of a user-friendly 3D visualization system for representing 2D partial differential equations (PDEs). Through collaborative efforts, we have integrated hardware and software components, enabling users to input their desired PDEs and boundary boundary conditions. Leveraging the computational power of the Raspberry Pi Pico microcontroller, numerical solutions to the equations are efficiently calculated. The design and implementation of a custom PCB have expanded the system's port capabilities, facilitating precise control over the height of sticks used for visualization. Incorporating ultrasonic sensors has enabled real-time feedback on stick height, ensuring accurate representation of curve features. Additionally, our system utilize a projector to project vibrant colors onto a stretchable canvas, spandex, yielding visually compelling visualization results. Overall, our accomplishment signifies a significant step forward in enhancing data comprehension and facilitating interactive exploration and analysis of complex systems' behavior in diverse scientific and engineering domains.

5.2 Uncertainties

While our project has achieved significant progress, it's important to acknowledge remaining uncertainties and potential challenges. One such uncertainty pertains to the attachment of the canvas to the top of the sticks. If the canvas is affixed too tightly, it may impede the stick's movement, limiting their ability to accurately represent the function surface. Conversely, if the canvas is not securely fixed to the sticks, there is a risk of it falling off, compromising the visualization process. Additionally, the large number of sticks necessitates a considerable amount of wiring, which could lead to potential instability in sensor feedback due to wire entanglement. To address these uncertainties. modifications to performance specifications could include exploring alternative attachment methods for canvas that balance stability and flexibility, as well as implementing cable management strategies to minimize wire interference and ensure reliable sensor feedback. By identifying and mitigating potential challenges, we can enhance the robustness and effectiveness of the visualization system.

5.3 Ethics Considerations

Prioritizing safety in the creation and application of any product is paramount. To ensure "the safety, health, and welfare of the public" as outlined in IEEE's ethical guidelines[6], we should strictly adhere to relevant regulations throughout the research and development phases, as well as inform users about the proper usage and to communicate the potential risks with misuse.

Furthermore, to be forthright and grounded in reality when making claims or estimates based on the data at hand[6], we should make it clear that the visualized differential equation solution is an approximation. Despite aiming to mirror real-life situations as closely as possible, these solutions cannot replace actual real-world solutions.

The essence of design lies in simplifying life and enhancing work efficiency. It's crucial for society to aim for respect, inclusivity, fairness, and balance, guaranteeing that everyone can access the necessary tools and resources for a rewarding life, free from discrimination related to race, religion, gender, disability, age, national origin, sexual orientation, gender identity, or expression[6]. Fundamentally, we are dedicated to providing individuals with an enhanced ability to interact with and learn from differential equations.

5.4 Future Work

There are several aspects that we can extend our projects for a wider application.

- **Real-time Visualization:** Implement real-time visualization capabilities so that the system can dynamically update and show changes in the function's behavior as it evolves over time. Users therefore can have a better understanding of how the PDEs changes over time. Overcoming challenges such as motor speed limitations, data transmission speed between PC and Raspberry Pi Pico, and projector response time will be crucial for achieving smooth real-time visualization.

- **Increase Stick Density for Higher Accuracy:** Enhance the accuracy of the visualization by increasing the number of sticks used to represent the function surface. This will require expanding the system's hardware capabilities, such as adding more ports to accommodate additional sticks.
- **Customizable Input and Parameters:** Allow user to input their own partial differential equations (PDEs) rather than choose from predefined ones. To achieve this, we need to explore more methods to solve different kind of PDEs accurately. For example, we can leverage the idea of neural networks to solve the equations [7]. Additionally, enable user to select specific regions of interest for visualization and fine-tune parameters to adjust the visualization according to their needs. Furthermore, consider extending the system's capabilities to visualize other types of 3D models beyond PDEs.
- **User Interface Enhancements:** Polish the user interface to improve usability and accessibility. Consider developing a dedicated website or more intuitive graphical user interface (GUI) for users to input differential equations, select visualization parameters and interact with the system more effectively. This can involve incorporating features such as tooltips, interactive tutorials, and visual aids to guide users through the process of using the system.

By addressing these aspects in future development, the 3D visualization system can become more versatile, user-friendly, and capable of providing valuable insights into complex systems' behavior across various domains.

References

[1] J. Wood, S. Kirschenbauer, J. Döllner, A. Lopes, and L. Bodum, "Using 3d in visualization," in *Exploring geovisualization*, Elsevier, 2005, pp. 293–312.

[2] A. R. Teyseyre and M. R. Campo, "An overview of 3d software visualization," *IEEE transactions on visualization and computer graphics*, vol. 15, no. 1, pp. 87–105, 2008.

[3] M. M. Shilleh. "Transfer data from raspberry pi pico to local computer." (2023), [Online]. Available: https://www.hackster.io/Shilleh/transfer-data-from-raspberry-pi-pico-to-local-computer-54ea9e.

[4] P. J. Roache, "Code verification by the method of manufactured solutions," *J. Fluids Eng.*, vol. 124, no. 1, pp. 4–10, 2002.

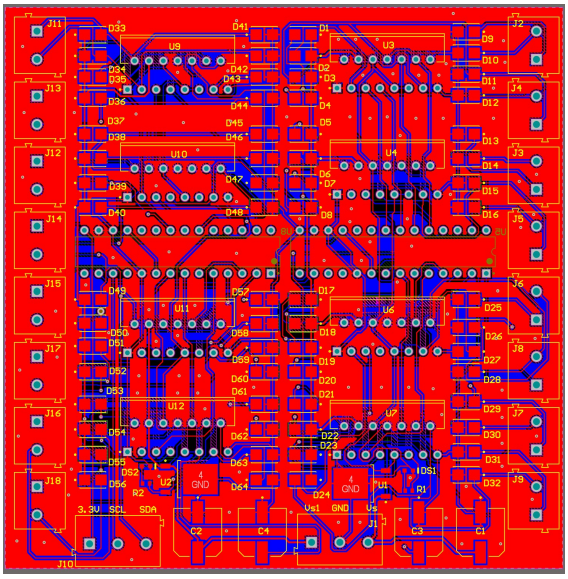
[5] U. E. Department. "Salary averages." (2024), [Online]. Available: Salary%20Averages (visited on 03/27/2024).

[6] IEEE. "Ieee code of ethics." (2020), [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html (visited on 03/05/2024).

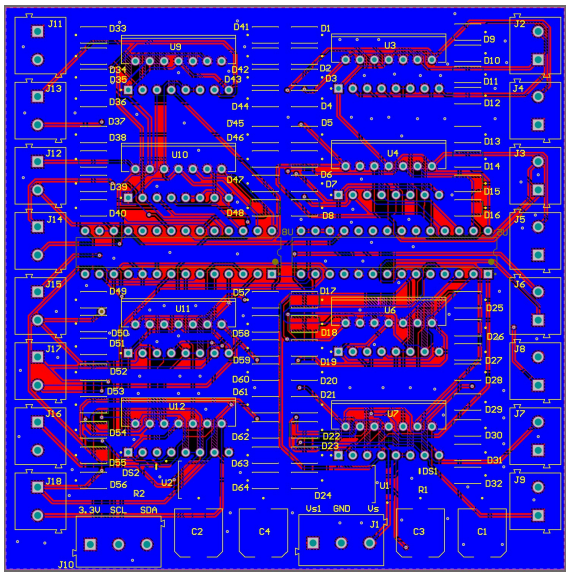
[7] A. Koryagin, R. Khudorozkov, and S. Tsimfer, "Pydens: A python framework for solving differential equations with neural networks," *arXiv preprint arXiv:1909.11544*, 2019.

Appendix A PCB Design Detail

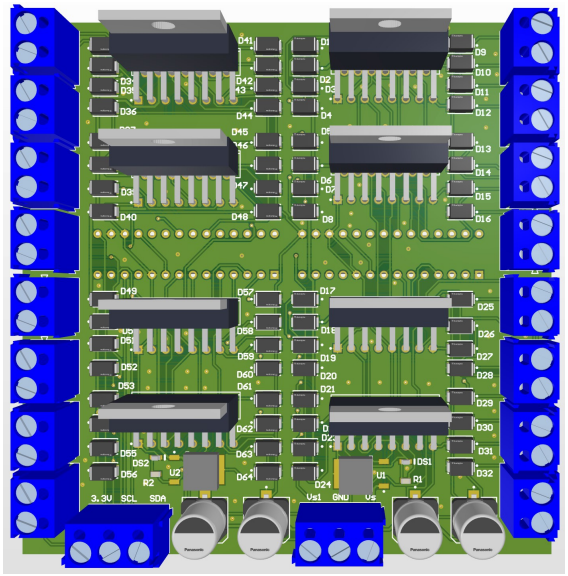
The top and bottom layers of the PCB in both 2D and 3D models are shown in Fig 19. The schematic of the PCB is shown in Fig 20.



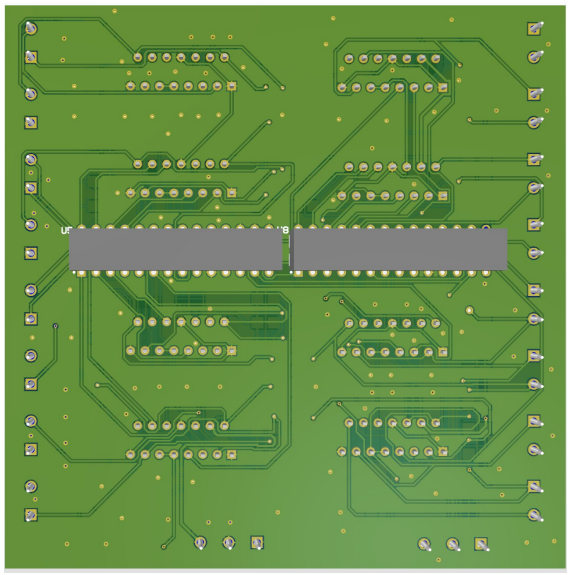
(a) Top layer of the PCB in 2D



(b) Bottom layer of the PCB in 2D



(c) Top view of the PCB in 3D



(d) Bottom view of the PCB in 3D

Figure 19: Top and bottom layers of the PCB in 2D and 3D

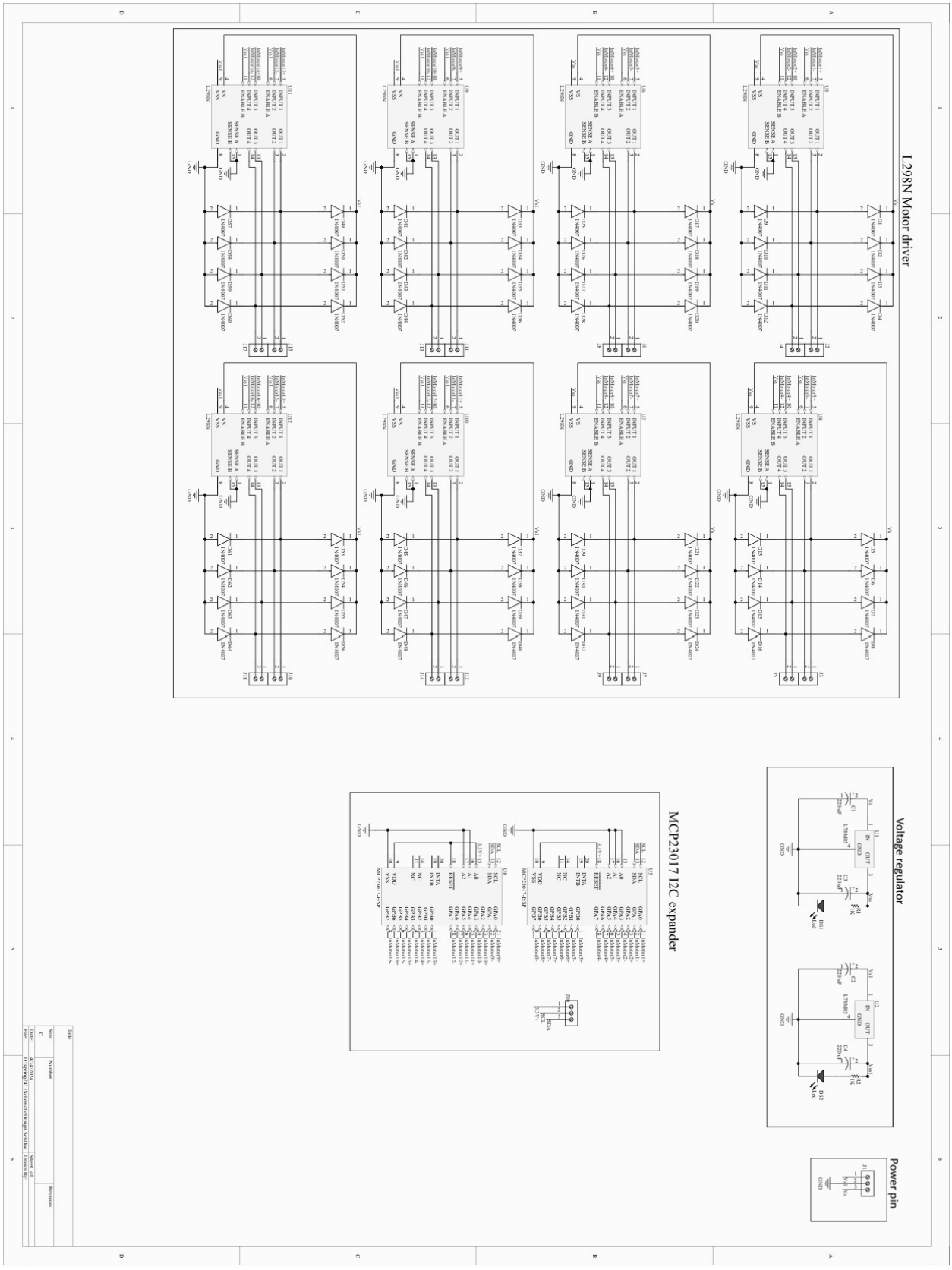


Figure 20: The overall schematic of the PCB

Appendix B User Interface Code

The following code shows the implementation of User Interface.

```
import serial
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp2d
import time

# Configure the serial connection

port = "/dev/cu.usbmodem1201"  # Adjust the port to match your setup

baudrate = 115200

serial_connection = serial.Serial(port, baudrate)

# Read and write data until the transfer is complete

# Prompt user for the range of data to send

print("Choose the partial differential equation type and boundary conditions:")
print("1. Heat equation -- Fixed temperature at boundaries (0 C  on left and bottom, 100 C  on right and top)")
print("2. Heat equation -- Linear boundaries (linear temperature gradient from 0 C  to 100 C  along each edge)")
print("3. Wave equation -- Dirichlet (zero values at boundaries)")
print("4. Wave equation -- Periodic (wrap-around boundary conditions)")
print("5. Wave equation -- Stochastic (add stochasticity to the boundary values)")
print("6. Laplace equation -- f(x) = sin(pi*x) at left boundary")
print("7. Laplace equation -- f(x) = x^2 at left boundary")
print("8. Laplace equation -- f(x) = exp(-x) at the left boundary")

while True:
    boundary_choice = input("Enter your choice (a number from 1 to 8): ")
    if boundary_choice.isdigit():  # Check if input is a digit
        boundary_choice = int(boundary_choice)
        if 1 <= boundary_choice <= 8:  # Check if input is within the specified range
            break  # Break the loop if input is valid
    print("Invalid input. Please enter a number from 1 to 8.")

# Read and write data until the transfer is complete
# for i in range(start, end + 1):
```

```
#     print(i)
#     serial_connection.write((str(i) + ',').encode())
#     time.sleep(0.01)
print(boundary_choice)
serial_connection.write((str(boundary_choice) + ',').encode())
print('write')


time.sleep(5)

# Open a file on your computer to write the received data
file_path = "color_info.txt"
destination_file = open(file_path, "wb")

# Read and write data until the transfer is complete
while True:
    data = serial_connection.read(128)
    print(data)
    destination_file.write(data)
    if (b"EOF" in data) or (b"EO" in data) or (b"E" in data):
        break


# Close the files and serial connection
destination_file.close()
serial_connection.close()

printed_array = ""
with open(file_path, "r") as file:
    for line in file:
        # Strip whitespace and newline characters
        line = line.strip()

        # Check if the line is "EOF", if so, stop reading
        if line == "EOF" or line == "EO" or line == "E":
            break

        # Append the line to the text_before_eof string
        printed_array += line  # Add newline character to preserve line
            breaks
```

Appendix C Stick and Sensor Code

The following code shows the implementation of Stick Class. This Class has two methods– measure and move. The first method is to measure the current position of the stick and the second is to control the stick to move up and down.

```
from MUX_open_close import open_channel, close_channel
from VL53L0X import VL53L0X
from time import sleep
import reverse
class Stick:
    def __init__(self,num,channel,mux_addr,doffset,std_value,pin,i2c):
        self.num=num
        self.channel=channel
        self.channel=channel
        self.mux_addr=mux_addr
        self.doffset=doffset
        self.std_value=std_value
        self.pin=pin
        self.i2c = i2c
        print("I am stick", self.num)

    def measure(self,flag,tof):
        if flag[self.num-1] == 0:
            stick_rt_1 = tof.ping()-self.doffset
            stick_rt_2 = tof.ping()-self.doffset
            stick_rt_3 = tof.ping()-self.doffset
            stick_rt_4 = tof.ping()-self.doffset
            stick_rt = round((stick_rt_1+stick_rt_2+stick_rt_3+
               stick_rt_4)/4)
            return stick_rt
        else:
            return self.std_value

    def motor_move(self,stick_rt,flag):
        if stick_rt<-5:
            reverse.reverse(self.pin,0)
        else:
            if  self.std_value>stick_rt+2:    #move upward
                self.pin[0].output(0)
                self.pin[1].output(1)
            elif self.std_value<stick_rt-2:  #move downward
                self.pin[0].output(1)
                self.pin[1].output(0)
            else:                            #stop moving
                self.pin[0].output(1)
                self.pin[1].output(1)
                flag[self.num-1] = 1
```

```
42                  return flag
```

Appendix D Main Control Flow Code

The following code shows the main control flow of the control system.

```
import MC
from machine import Pin, I2C
from time import time_ns,sleep
from Stick_Unit import Stick
from MUX_open_close import open_channel, close_channel
from VL53L0X import VL53L0X

RUN_MOTOR = 1

i2c1 = I2C(1, scl=Pin(19), sda=Pin(18), freq = 10000)
print("device:", i2c1.scan())
mcp0 = MC.MCP23017(i2c1, 0x20)
mcp1 = MC.MCP23017(i2c1, 0x21)

i2c_mux1 = I2C(id=0,scl=Pin(13),sda=Pin(12),freq=400000)
for dev in i2c_mux1.scan():
    print("mux dev: ", hex(dev))
flag_finish = [0]*16

stick1 = Stick(1,3,0x71,46,30,[mcp1[4],mcp1[5]],i2c_mux1)
stick2 = Stick(2,6,0x71,44,30,[mcp1[10],mcp1[11]],i2c_mux1)
stick3 = Stick(3,3,0x70,111,30,[mcp0[8],mcp0[9]],i2c_mux1)
stick4 = Stick(4,7,0x71,35,30,[mcp0[4],mcp0[5]],i2c_mux1)
stick5 = Stick(5,8,0x71,53,30,[mcp1[6],mcp1[7]],i2c_mux1)
stick6 = Stick(6,4,0x70,60,30,[mcp1[8],mcp1[9]],i2c_mux1)
stick7 = Stick(7,5,0x70,42,30,[mcp0[10],mcp0[11]],i2c_mux1)
stick8 = Stick(8,6,0x70,89,30,[mcp0[12],mcp0[13]],i2c_mux1)
stick9 = Stick(9,7,0x70,47,30,[mcp0[14],mcp0[15]],i2c_mux1)
stick10 = Stick(10,4,0x71,43,30,[mcp0[2],mcp0[3]],i2c_mux1)
stick11 = Stick(11,1,0x70,51,30,[mcp1[12],mcp1[13]],i2c_mux1)
stick12 = Stick(12,8,0x70,56,30,[mcp0[6],mcp0[7]],i2c_mux1)
stick13 = Stick(13,1,0x71,56,30,[mcp1[2],mcp1[3]],i2c_mux1)
stick14 = Stick(14,5,0x71,54,30,[mcp1[14],mcp1[15]],i2c_mux1)
stick15 = Stick(15,2,0x71,57,30,[mcp1[0],mcp1[1]],i2c_mux1)
stick16 = Stick(16,2,0x70,55,30,[mcp0[0],mcp0[1]],i2c_mux1)


sticks=[stick1,stick2,stick3,stick4,stick5,stick6,stick7,stick8,stick9,
   stick10, stick11,stick12,stick13,stick14,stick15,stick16]
tofs=[]
num=len(sticks)
sticks_std= [40]*num


```

```
for i, stick in enumerate(sticks):
    stick.std_value = sticks_std[i]

sticks_pre_rt=[0]*num
sticks_rt = []

for i in range(0,num):
    open_channel(sticks[i].channel,sticks[i].mux_addr,sticks[i].i2c)
    tofs.append(VL53L0X(sticks[i].i2c))
    close_channel(sticks[i].channel,sticks[i].mux_addr,sticks[i].i2c)
print(tofs)


for i, stick in enumerate(sticks):
    open_channel(sticks[i].channel,sticks[i].mux_addr,sticks[i].i2c)
    sticks_rt.append(stick.measure(flag_finish,tofs[i]))
    close_channel(sticks[i].channel,sticks[i].mux_addr,sticks[i].i2c)
    print(sticks_rt)

while(True):
    for i in range(0,num):
        open_channel(sticks[i].channel,sticks[i].mux_addr,sticks[i].i2c
            )
        sticks_rt[i] = sticks[i].measure(flag_finish,tofs[i])
        close_channel(sticks[i].channel,sticks[i].mux_addr,sticks[i].
            i2c)
        if RUN_MOTOR:
            sticks[i].motor_move(sticks_rt[i],flag_finish)
    print("sticks real-time height:",sticks_rt)
    print("sticks std height:",sticks_std)
    sleep(0.1)
    if all(value == 1 for value in flag_finish):
        break
#
```